

# Routing in Polygons with Holes

Matias Korman<sup>\*</sup>      Wolfgang Mulzer<sup>†</sup>      André van Renssen<sup>‡,§</sup>  
 Marcel Roeloffzen<sup>‡,§</sup>      Paul Seiferth<sup>†</sup>      Yannik Stein<sup>†</sup>      Birgit Vogtenhuber<sup>¶</sup>  
 Max Willert<sup>†</sup>

## Abstract

Sending a message through an unknown network is a difficult problem. In this paper, we consider the case in which, during a preprocessing phase, we assign a *label* and a *routing table* to each node. The routing strategy must then decide where to send the package using only the label of the target node and the routing table of the node the message is currently at.

In this paper, we present the first routing scheme for the particular case in which the network is defined by the *visibility graph* of a polygon with  $n$  vertices and  $h$  holes. For any  $\varepsilon > 0$  the routing scheme provides stretch at most  $1 + \varepsilon$ . The labels have  $\mathcal{O}(\log n)$  bits and the corresponding routing tables are of size  $\mathcal{O}(\varepsilon^{-1}(h+1)\log n)$ . The preprocessing time is  $\mathcal{O}(n^2 \log n + (h+1)n^2 + \varepsilon^{-1}(h+1)n)$  and can be improved to  $\mathcal{O}(n^2 + \varepsilon^{-1}n)$  for simple polygons.

## 1 Introduction

Routing is a crucial problem in distributed graph algorithms [9, 18]. Given a graph  $G$ , we would like to send a data package from one location to another. There are two different properties that any routing scheme should satisfy. First, the routing algorithm should be *local*, i.e., it should only use information that is stored in the data package and the preprocessed memory of the current node of the graph. Ideally, the routing scheme should also be *efficient*, meaning that the data package does not travel much longer than necessary.

There is an obvious way to solve a routing problem: for each node  $p$  of  $G$ , we store the complete shortest path tree of  $p$  in its memory. Thus, given a target destination we can send the message one step towards the destination in the shortest path, and repeat this until we reach the destination. Unfortunately, this method requires that each node stores the entire topology of  $G$ , which could be prohibitive. The two goals are somehow in conflict: on the one hand, the routing scheme should yield paths that are as short as possible, but on the other hand, we would like to store only a small amount of routing information in each node's local memory.

Thorup and Zwick introduced the notion of a *distance oracle* [26]. Given a graph  $G$ , the goal is to produce a compact data structure that can quickly answer distance queries between any pair of nodes in the graph. We can think of routing schemes as distributed implementations of distance oracles [20]. The graph corresponds to the distributed network. A distributed algorithm runs on the processors – the nodes of the graph. Each node has a memory and a data package contains the name of the destination and perhaps some additional information for the routing scheme. If a node receives a message, it checks whether it is the message's

<sup>\*</sup>Tohoku University, Sendai, Japan. Partially supported by the ELC project (MEXT KAKENHI No. 12H00855 and 15H02665).

<sup>†</sup>Department of Computer Science, Freie Universität Berlin, Germany

<sup>‡</sup>National Institute of Informatics (NII), Tokyo, Japan.

<sup>§</sup>JST, ERATO, Kawarabayashi Large Graph Project.

<sup>¶</sup>Institute for Software Technology, Graz University of Technology, Graz, Austria.

destination. Otherwise, it uses the additional information and its own routing table to choose the link to which it forwards the message. The *stretch* of a routing scheme is the worst possible ratio between the travelled distance and the shortest path distance.

The problem has been well-studied for general graphs for a long time [1, 3, 5–7, 19, 20]. One of the most recent results is from 2016 from Roditty and Tov [20], who developed a routing scheme for general graphs  $G$  with  $n$  vertices and  $m$  edges. Their scheme provides a poly-logarithmic header size and routes a message from  $p$  to  $q$  on a path with length  $\mathcal{O}(k\Delta + m^{1/k})$ , where  $\Delta$  is the distance of the shortest path between  $p$  and  $q$  and  $k > 2$  an integer. Their routing tables use  $mn^{\mathcal{O}(1/\sqrt{\log n})}$  total space. It is known that in general graphs any efficient routing scheme needs to store  $\Omega(n^c)$  bits per node, for some  $c > 0$  [18]. Thus, it is useful to ask whether there are better algorithms for specialized graph classes. For instance, there are routing schemes for trees that follow the shortest path and require at most  $\mathcal{O}(\log n)$  bits at each node [8, 21, 25]. Additionally, in planar graphs it is possible for any  $\varepsilon > 0$  to find a routing scheme with a poly-logarithmic number of bits in the routing tables and a stretch of  $1 + \varepsilon$  [24].

Our graph class of interest is the visibility graph of a polygon with an arbitrary number of holes. Two vertices in a polygon  $P$  with  $h$  holes and  $n$  vertices are connected by an edge if and only if they can see each other, i.e., the line segment between the two vertices is contained in the closed interior of  $P$ . The problem of computing a shortest path between two vertices in a polygon has been well-studied in computational geometry [2, 4, 10, 11, 13–17, 22, 23, 27]. Nevertheless, to the best of our knowledge, up to now there have not been any routing schemes for visibility graphs of polygons. Thus, we present the first such routing scheme. For any  $\varepsilon > 0$  the routing scheme needs at most  $\mathcal{O}(\varepsilon^{-1}(h+1)\log n)$  bits in the routing table and produces a routing path with stretch  $(1 + \varepsilon)$ .

## 2 Preliminaries

Let  $G = (V, E)$  be an *undirected*, *connected* and *simple* graph. In our model this graph  $G$  is embedded in the Euclidean plane: a *node*  $p = (p_x, p_y) \in V$  corresponds to a point in the plane and an edge  $\{p, q\} \in E$  is represented by the line segment  $\overline{pq}$ . We denote by  $|\overline{pq}|$  the Euclidean distance between the points  $p$  and  $q$  and call  $|\overline{pq}|$  the *length* of the corresponding edge. The length of a shortest path connecting two points  $p, q \in V$  is denoted by  $d(p, q)$ .

Now we define a *routing scheme* for  $G$ . Each physical node has a name tag, called *label*, that identifies the physical node in the network, as well as some information stored in the memory of the physical node, called *routing table*. The routing scheme works as follows: starting at a node  $p \in V$  we use the information of  $p$ 's routing table and the target's label. This behaviour is modeled by a *routing function*. The routing function computes a new node adjacent to  $p$  where the data package is forwarded to. This process is repeated until the data package reaches its destination.

**Definition 2.1** (Routing scheme). *A routing scheme  $\mathcal{R} = (l, \rho, f)$  of a graph  $G$  consists of the following elements: a label  $l(p) \in \{0, 1\}^*$  and a routing table  $\rho(p) \in \{0, 1\}^*$  for each node  $p \in V$  as well as a global routing function  $f: V \times l(V) \rightarrow V$ .*

We note that minor alterations of the routing scheme have been proposed in the literature [12, 20, 28]. For example, sometimes it is allowed to store additional information in the *header* of a data package (this header is part of the data package and will travel along with the message). Similarly, sometimes it is allowed to use an intermediate target label. This is helpful for encoding extra information and allowing recursive routing schemes. In this paper we do not use either of these tools.

Given the routing scheme, a start node  $p \in V$  and a destination  $q \in V$ , we send the message by repeatedly applying function  $f$ . That is, the sequence of nodes visited is  $p_0 = p$  and  $p_i = f(p_{i-1}, l(q))$  for  $i \geq 1$ . Naturally, we are interested in routing schemes for which the

message always arrives at its destination. More formally, we say that a routing scheme  $\mathcal{R}$  is *correct* if for any  $p, q \in V$  there exists a  $k = k(p, q) \geq 0$  such that  $p_k = q$  (and  $p_i \neq q$  for  $i < k$ ). We call  $p_0, p_1, \dots, p_k$  the *routing path* between  $p$  and  $q$ . The *routing distance* between  $p$  and  $q$  is defined as  $d_\rho(p, q) = \sum_{i=1}^k |\overline{p_{i-1}p_i}|$ .

The quality of the routing scheme is measured by several parameters:

- the *label size*  $L(n) = \max_{|V|=n} \max_{p \in V} |l(p)|$ ,
- the *table size*  $T(n) = \max_{|V|=n} \max_{p \in V} |\rho(p)|$ ,
- the *stretch*  $\zeta(n) = \max_{|V|=n} \max_{p \neq q \in V} d_\rho(p, q)/d(p, q)$  and
- the preprocessing time.

Let  $P$  be a polygon with  $n$  vertices and  $h$  holes. The *boundary*  $\partial P$  of the polygon consists of  $h+1$  pairwise disjoint simple closed polygonal chains: one outer boundary and  $h$  hole boundaries. We denote the *interior* of  $P$  by  $\text{int } P$ , i.e.,  $\text{int } P = P \setminus \partial P$ . Note that we make no general position assumption of the vertices of  $P$ . Let  $n_0$  be the number of vertices on the outer boundary of  $P$  and let  $n_i$ ,  $1 \leq i \leq h$  be the number of vertices on the  $i$ -th hole of  $P$ , respectively. Each vertex is assigned two indices  $0 \leq i \leq h$  and  $0 \leq j \leq n_i - 1$ , in the following manner:  $p_{i,j}$  is the  $j$ -th vertex on the  $i$ -th boundary of  $P$ , where the second index  $j$  increases clockwise along the according boundary.

For  $p$  and  $q$  in  $P$  we say that  $p$  and  $q$  can *see each other* if and only if  $\overline{pq} \subset P$ . Remark that  $p$  and  $q$  can see each other even if the line segment  $\overline{pq}$  hits  $\partial P$ . The visibility graph of  $P$ , denoted by  $\text{VG}(P)$  is a graph whose vertices are the vertices of  $P$  and contains an edge between two vertices if and only if they see each other in  $P$ .

We show the following main theorem:

**Theorem 2.1.** *For any  $\varepsilon > 0$  and any polygon  $P$  with  $n$  vertices and  $h$  holes we can preprocess  $P$  and construct a routing scheme for  $\text{VG}(P)$  with  $\zeta(n) = 1 + \varepsilon$ ,  $L(n) = \mathcal{O}(\log n)$  and  $T(n) = \mathcal{O}(\varepsilon^{-1}(h+1) \log n)$ . The preprocessing time is  $\mathcal{O}(n^2 \log n + (h+1)n^2 + \varepsilon^{-1}(h+1)n)$  and can be improved to  $\mathcal{O}(n^2 + \varepsilon^{-1}n)$  for simple polygons.*

### 3 Cones in Polygons

Let  $P$  be a polygon with  $h$  holes and  $n$  vertices. Furthermore, let  $t > 2$  be a parameter. We use a technique of Yao [29] to subdivide the visibility polygon of a site into rays and cones with a certain apex angle. This partition is then used for our routing scheme.

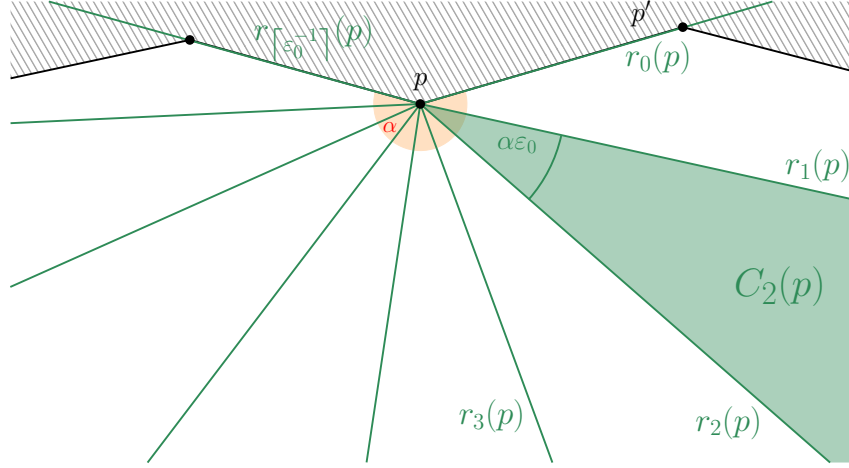
Let  $p$  be a vertex in the polygon and  $p'$  the clockwise (counterclockwise) next vertex on the outer boundary (the boundary of a hole). We denote with  $\mathbf{r}$  the *ray* emanating from  $p$  through  $p'$ . Next, we rotate this ray by certain angles. Let  $\alpha$  be the inner angle at  $p$  and  $\varepsilon_0 := \frac{2\pi}{\alpha t}$ . We set

$$r_i(p) := \mathbf{r} \text{ rotated clockwise by angle } \alpha \cdot \min(i \cdot \varepsilon_0, 1)$$

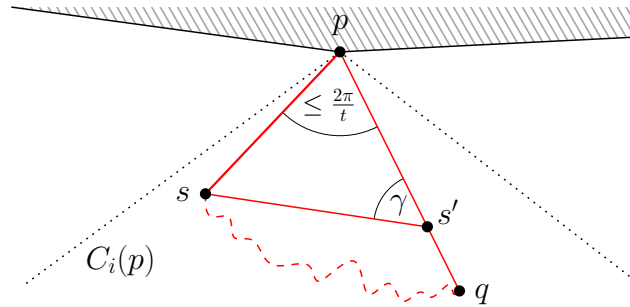
for  $0 \leq i \leq \lceil \varepsilon_0^{-1} \rceil$ . Using the rays, we define the *cones* of  $p$ .  $C_i(p)$  is the closed cone with apex  $p$  and boundary  $r_{i-1}(p) \cup r_i(p)$  that does not contain any ray  $r_j(p)$  in its interior (see Figure 1). Furthermore,  $\mathcal{C}(p)$  is the set of all cones with apex  $p$  whose boundaries emanate into  $P$ , that is,  $\mathcal{C}(p) = \{C_i(p) \mid 1 \leq i \leq \lceil \varepsilon_0^{-1} \rceil\}$ . Observe that the apex angle of each cone is at most  $\alpha \varepsilon_0 = 2\pi/t$  (and hence each cone is convex).

**Lemma 3.1.** *Let  $p$  be a vertex in  $P$  and  $\{p, q\}$  an edge of  $\text{VG}(P)$  in the cone  $C_i(p)$ . Furthermore, let  $s$  be the visible vertex in  $C_i(p)$  that is closest to  $p$ . Then the following inequality holds:*

$$d(s, q) \leq |\overline{pq}| - \left(1 - 2 \sin \frac{\pi}{t}\right) |\overline{ps}|.$$



**Fig. 1:** The cones and rays of a vertex  $p$  with inner angle  $\alpha$ .



**Fig. 2:** Illustration of Lemma 3.1. The points  $s$  and  $s'$  have the same distance to  $p$ . The dashed line represents the shortest path from  $s$  to  $q$ .

*Proof.* First of all, let  $s'$  be the point on the line segment  $\overline{pq}$  such that  $|\overline{ps'}| = |\overline{ps}|$  (see Figure 2). Since  $p$  can see  $q$ ,  $s'$  can also see  $q$ . Furthermore, by construction  $s$  can see  $s'$ , because if the line segment  $\overline{ss'}$  intersected the boundary of  $P$ , we could either find a vertex  $s''$  contained in  $\Delta(p, s, s')$  that can see  $p$  with  $|\overline{ps}| > |\overline{ps''}|$  or the boundary of  $P$  would hit at least one of the line segments  $\overline{ps}$  and  $\overline{pq}$ , by this contradicting that  $p$  can see  $s$  and  $q$ . Now the triangle inequality yields  $d(s, q) \leq |\overline{ss'}| + |\overline{s'q}|$ . Let  $\beta$  be the angle at  $p$  in the triangle  $\Delta(p, s, s')$ . This angle is at most  $2\pi/t$ . Thus, the angle at  $s'$  is  $\gamma = \pi/2 - \beta/2$ . Using the sine law and  $\sin 2x = 2 \sin x \cos x$  we get

$$|\overline{ss'}| = |\overline{ps}| \cdot \frac{\sin \beta}{\sin \gamma} = |\overline{ps}| \cdot \frac{\sin \beta}{\sin \left( \frac{\pi}{2} - \frac{\beta}{2} \right)} = |\overline{ps}| \cdot \frac{2 \sin \frac{\beta}{2} \cos \frac{\beta}{2}}{\cos \frac{\beta}{2}} \leq 2|\overline{ps}| \sin \frac{\pi}{t}.$$

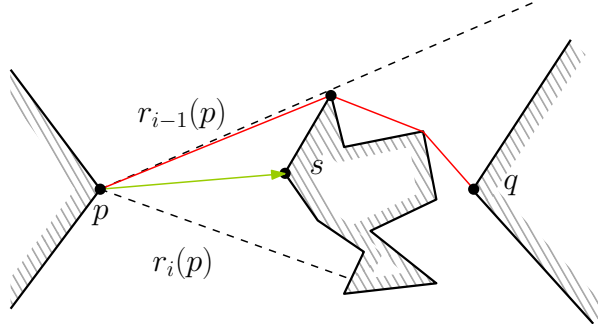
Furthermore, we have  $|\overline{s'q}| = |\overline{pq}| - |\overline{ps'}| = |\overline{pq}| - |\overline{ps}|$  and thus, the triangle inequality gives

$$d(s, q) \leq 2|\overline{ps}| \sin \frac{\pi}{t} + |\overline{pq}| - |\overline{ps}| = |\overline{pq}| - \left( 1 - 2 \sin \frac{\pi}{t} \right) |\overline{ps}|.$$

□

## 4 The Routing Scheme

Let  $\varepsilon > 0$  and  $P$  be a polygon with  $n$  vertices and  $h$  holes. We describe a routing scheme for  $\text{VG}(P)$  with stretch factor  $1 + \varepsilon$ . The idea is to compute for each vertex  $p$  the corresponding set of cones  $\mathcal{C}(p)$  and to save a certain set of indices for each cone in the routing table of  $p$ . These sets of indices form intervals. If an interval of a cone  $C_i(p)$  contains the target vertex  $q$  we proceed to the nearest neighbour in  $C_i(p)$  (see Figure 3). We will see that this results in a path with small stretch.



**Fig. 3:** The idea of the routing scheme. The first edge on a shortest path from  $p$  to  $q$  (red) is contained in  $C_i(p)$ . The routing algorithm will route the package from  $p$  to  $s$  (green), the closest vertex to  $p$  in  $C_i(p)$ .

For the preprocessing we first compute the label  $l(p)$  of each vertex  $p$ . Each such label consists of the two indices of the numbering in  $P$ . The first part of  $l(p)$  represents the index of one of the  $h + 1$  components of the boundary, whereas the second part describes the index of the vertex on the fixed boundary component. All the labels are distinct binary strings and have length  $\mathcal{O}(\log n)$ .

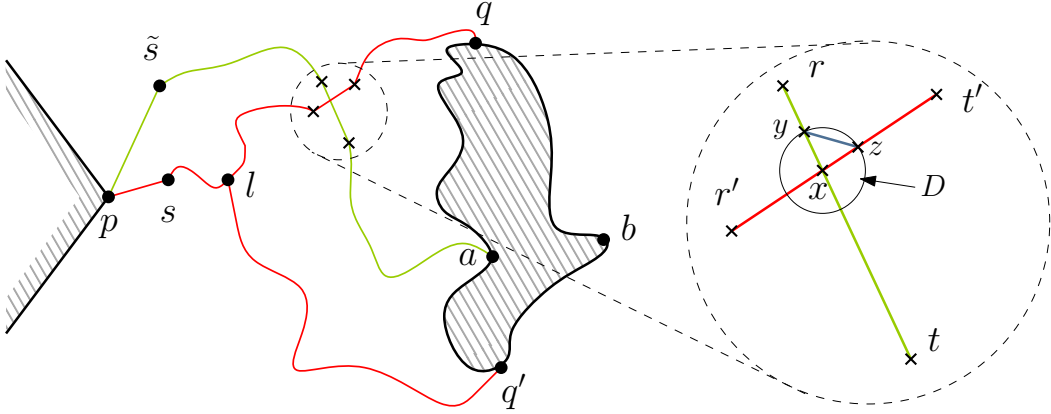
For the routing table we do the following: We compute the shortest path tree  $T_p$  starting from  $p$ . Let  $T_p(s)$  be a subtree of  $T_p$  with root  $s$  and denote the set of all vertices of the  $i$ -th hole in  $T_p(s)$  by  $I_s(i)$ . We prove the following lemma.

**Lemma 4.1.** *Let  $p$  be a vertex in  $P$  and let  $e = (p, s)$  be an edge in  $T_p$ . Then the indices of the vertices in  $I_s(i)$  form an interval. Furthermore, let  $f = (p, s')$  be another edge in  $T_p$ , such that  $e$  and  $f$  are consecutive edges in the cyclic order around  $p$  in  $T_p$ . Then the indices of the vertices in  $I_s(i) \cup I_{s'}(i)$  are again an interval.*

*Proof.* For the first part of the lemma, consider two distinct vertices  $q, q' \in I_s(i)$ , with lowest common ancestor  $l$  in the shortest path tree. We assume that  $q$  and  $q'$  are not contained in an interval.

Thus, we can find two vertices  $a, b \notin I_s(i)$ , that lie on different polygonal chains with endpoints  $q$  and  $q'$  induced by the boundary of the  $i$ -th hole (see Figure 4). Then there has to be a neighbour  $\tilde{s}$  of  $p$  such that the following holds:

- $\tilde{s}$  lies on the shortest path from  $p$  to  $a$  or  $p$  to  $b$  in  $T_p$  and
- the shortest path from  $\tilde{s}$  to  $a$  or the shortest path from  $\tilde{s}$  to  $b$  first crosses the shortest path from  $l$  to  $q$  or the shortest path from  $l$  to  $q'$  in  $T_p$ .



**Fig. 4:** The green line is the shortest path from  $p$  to  $a$ , whereas the red paths are the “shortest” paths from  $p$  to  $q$  and  $q'$ . The blue line segment gives a shortcut from  $r$  to  $t'$ .

We consider the first case and derive a contradiction. The other cases are analogous. In this case,  $\tilde{s}$  lies on the shortest path  $\pi$  from  $p$  to  $a$  and it crosses the shortest path  $\pi'$  from  $l$  to  $q$ . Let  $x$  be the first intersection of  $\pi$  and  $\pi'$ . The intersection  $x$  is not a vertex of  $P$ , because otherwise  $T_p$  would have a cycle.

Consider the two directed edges  $(r, t)$  and  $(r', t')$  in  $T_p$  intersecting at  $x$ , where  $r$  and  $t$  lie on  $\pi$  and  $(r', t')$  on  $\pi'$ . Again, we have two cases and we discuss only the first one:  $d(p, r) + |\overline{rx}| \leq d(p, r') + |\overline{r'x}|$  and vice versa. Since  $(r, t)$  and  $(r', t')$  are edges in  $T_p$  and  $x \in \text{int } P$ , these edges hit the boundary of  $P$  only in their endpoints. Thus, we can find a  $\delta > 0$  such that the disk  $D$  with center  $x$  and radius  $\delta$  is contained in  $P$ . Now consider the intersection  $y$  of  $\overline{rx}$  and the boundary of  $D$  and the intersection  $z$  of  $\overline{t'x}$  and the boundary of  $D$  (see Figure 4). We have  $\overline{yz} \subset D \subset P$  and the triangle inequality yields  $|\overline{xy}| + |\overline{xz}| > |\overline{yz}|$ . Thus, the path  $ryzt'$  is a shortcut from  $s$  to  $t'$ . This shortcut gives us a shorter path from  $p$  to  $q$  using the vertex  $\tilde{s}$ . Hence, we have  $q \in I_{\tilde{s}}(i) \neq I_s(i)$  which contradicts the assumption.

For the second part of the lemma, the main argument remains the same. We consider two vertices  $q, q' \in I_s(i) \cup I_{s'}(i)$  and two vertices  $a, b \notin I_s(i) \cup I_{s'}(i)$  that lie on two different polygonal chains with endpoints  $q$  and  $q'$  induced by the boundary of the  $i$ -th hole. Again, there has to be a neighbour  $\tilde{s}$  of  $p$ . This neighbour satisfies the conditions above, because it is not contained in the cone with apex  $p$  spanned by the edges  $e$  and  $f$ . Then we again derive a contradiction and hence the claim follows.  $\square$

Lemma 4.1 gives the main idea for the routing table. We subdivide the part of the polygon around  $p$  into cones with a certain angle. Thus, we set

$$t := \pi / \arcsin \left( \frac{1}{2(1 + \varepsilon^{-1})} \right)$$

and use the subdivision described in Section 3. This subdivision provides a set  $\mathcal{C}(p)$  with a certain number of cones. The following lemma specifies this number.

**Lemma 4.2.** *We have  $t \leq 2\pi(1 + \varepsilon^{-1})$ .*

*Proof.* We have  $\sin x \leq x$  for  $0 < x < 1/2$ . Now for  $z = 1/x > 2$ , we obtain the inequality  $\sin(1/z) \leq 1/z$  and by the monotonicity of the sine function in the interval  $(0, 1/2)$  we have  $\arcsin(1/z) \geq 1/z$ . Thus, we can substitute  $z = 2(1 + \varepsilon^{-1})$  to obtain the desired inequality.  $\square$

Lemma 4.2 gives  $|\mathcal{C}(p)| \in \mathcal{O}(\varepsilon^{-1})$ . For each cone  $C_j(p) \in \mathcal{C}(p)$  and each hole we collect all the indices  $I_s(i)$  with  $s \in C_j(p)$ . By Lemma 4.1 these indices form an interval. This interval contains all indices of vertices whose shortest path starts in the cone  $C_j(p)$ . To use less space, we save for each cone the intervals of  $h + 1$  holes and the label of one vertex that has the shortest distance to  $p$  in  $C_j(p)$ . An interval needs  $\mathcal{O}(\log n)$  bits, because we store only the boundaries of the interval. Furthermore, in each cone we have  $\mathcal{O}(h)$  intervals and one label of size  $\mathcal{O}(\log n)$ . Thus, the size of the routing table is  $\mathcal{O}(\varepsilon^{-1}(h + 1) \log n)$ .

The routing function is quite simple. Starting at a vertex  $p$ , we search the label  $l(q)$  of the target vertex  $q$  in the routing table  $\rho(p)$ . The search of the label gives a cone  $C_j(p)$  and the label of the corresponding nearest neighbour of  $p$  in  $C_j(p)$ . We then travel from  $p$  to this neighbour (see Figure 3).

## 5 Analysis

In this section we prove the stretch factor of our routing scheme and give an upper bound on the preprocessing time. Let  $1 + \varepsilon$ ,  $\varepsilon > 0$ , be the desired stretch factor and let  $t$  be as in the previous section. First, we show that during each routing step the distance to the target vertex decreases.

**Lemma 5.1.** *Let  $p$  and  $q$  be two vertices in  $P$ . Furthermore, let  $s$  be the next vertex computed by the routing scheme that routes a data package from  $p$  to  $q$ . Then we have  $d(s, q) \leq d(p, q) - |\overline{ps}|/(1 + \varepsilon)$ .*

*Proof.* First, we look at the routing table of  $p$  to find the label  $l(q)$ . This query supplies a cone and the label of the corresponding vertex  $s$  that the package is routed to next. By construction, we know that the next vertex  $q'$  on the shortest path from  $p$  to  $q$  is contained in the same cone as  $s$ . Hence, by the triangle inequality and Lemma 3.1, we obtain

$$\begin{aligned} d(s, q) &\leq d(s, q') + d(q', q) \\ &\leq |\overline{pq'}| - \left(1 - 2 \sin \frac{\pi}{t}\right) |\overline{ps}| + d(q', q) \\ &= d(p, q) - \left(1 - 2 \sin \frac{\pi}{t}\right) |\overline{ps}| \\ &= d(p, q) - \left(1 - \frac{1}{1 + \varepsilon^{-1}}\right) |\overline{ps}| && \text{(by the choice of } t) \\ &= d(p, q) - |\overline{ps}|/(1 + \varepsilon). \end{aligned}$$

$\square$

Lemma 5.1 immediately implies the correctness of the routing scheme: Because the distance to the target vertex  $q$  strictly decreases in each step and because there is a finite number of vertices in the polygon, we can find a  $k = k(p, q) \leq n$  such that  $\pi = p_0 p_1 \dots p_k$  is the routing path with  $p = p_0$  and  $q = p_k$ . Using this, we can now bound the stretch of the routing scheme.

**Lemma 5.2.** *Let  $p$  and  $q$  be two vertices of  $P$ . Then we have  $d_\rho(p, q) \leq (1 + \varepsilon)d(p, q)$ .*

*Proof.* Let  $\pi = p_0 p_1 \dots p_k$  be the path from  $p = p_0$  to  $q = p_k$  computed by the routing scheme. By Lemma 5.1 we have  $d(p_{i+1}, q) \leq d(p_i, q) - |\overline{p_i p_{i+1}}|/(1 + \varepsilon)$ . Thus, we have

$$\begin{aligned} d_\rho(p, q) &= \sum_{i=0}^{k-1} |\overline{p_i p_{i+1}}| \\ &\leq (1 + \varepsilon) \sum_{i=0}^{k-1} (d(p_i, q) - d(p_{i+1}, q)) \\ &= (1 + \varepsilon) (d(p_0, q) - d(p_k, q)) \\ &= (1 + \varepsilon) d(p, q) \end{aligned}$$

since  $p_0 = p$  and  $p_k = q$ . This finishes the proof for the bound of the stretch.  $\square$

Finally, we discuss the details of the preprocessing and its time complexity.

**Lemma 5.3.** *The preprocessing time for the described routing scheme is  $\mathcal{O}(n^2 \log n + (h + 1)n^2 + \varepsilon^{-1}(h + 1)n)$ .*

*Proof.* Let  $p$  be a vertex of  $P$ . First of all, we compute the shortest path tree  $T_p$ . Using the algorithm of Hershberger and Suri [11], this can be done in time  $\mathcal{O}(n \log n)$ . Now we make a post-order traversal of  $T_p$  to compute the intervals for each child of  $p$ . Given a node  $q$ , the post-order traversal provides at most  $h + 1$  different intervals. For each hole we compute the union of the intervals among the children. Lemma 4.1 shows that the union of these intervals is again an interval. This union can be done in time  $\mathcal{O}((h + 1) \text{outdeg}(q))$ , where  $\text{outdeg}(q)$  is the number of children of  $q$  in  $T_p$ . In total this post-order traversal needs  $\mathcal{O}((h + 1)n)$  time.

Let  $q_1 \dots q_k$  be the children of  $p$  and let  $\alpha_1 \dots \alpha_k$  be the corresponding angles spanned by the ray  $r_0(p)$  and the edge  $(p, q_j)$ . By construction, the  $q_j$ 's are sorted by increasing angle  $\alpha_j$ . Moreover, we insert into this sorted sequence  $L$  the rays  $r_i(p)$ . By Lemma 4.2 the sequence  $L$  contains  $\mathcal{O}(\varepsilon^{-1} + \text{outdeg}(p))$  elements. Next, we scan through  $L$ . Between two rays  $r_{i-1}(p)$  and  $r_i(p)$  we join all the corresponding intervals. Again by Lemma 4.1 these unions are intervals. Furthermore, we compute the point that is closest to  $p$ . Finally, we store the label of the closest point and the  $(h + 1)$  intervals in the  $i$ -th row of the routing table  $\rho(p)$ . This last step takes time  $\mathcal{O}((h + 1)(\varepsilon^{-1} + \text{outdeg}(p))) = \mathcal{O}((h + 1)\varepsilon^{-1} + (h + 1)n)$ .

Thus, the computation time for one point is  $\mathcal{O}(n \log n + (h + 1)n + (h + 1)\varepsilon^{-1})$  and we need preprocessing time  $\mathcal{O}(n^2 \log n + (h + 1)n^2 + \varepsilon^{-1}(h + 1)n)$ .  $\square$

Combining the last two lemmas and the discussion in Section 4 we obtain the following theorem.

**Theorem 5.4.** *Let  $P$  be a polygon with  $n$  vertices and  $h$  holes. For any  $\varepsilon > 0$  we can preprocess  $P$  and construct a routing scheme for  $\text{VG}(P)$  with labels of size  $\mathcal{O}(\log n)$  bits and routing tables of size  $\mathcal{O}(\varepsilon^{-1}(h + 1) \log n)$  bits. For any two sites  $p, q \in P$ , the scheme produces a routing path with stretch at most  $1 + \varepsilon$ . The preprocessing time is  $\mathcal{O}(n^2 \log n + (h + 1)n^2 + \varepsilon^{-1}(h + 1)n)$ .*

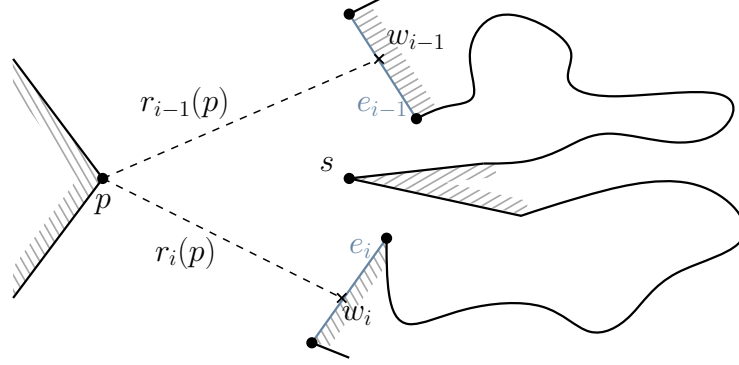
## 6 Improvement for simple Polygons

Let  $P$  be a simple polygon with  $n$  vertices. Again, let  $1 + \varepsilon$ ,  $\varepsilon > 0$ , be the stretch factor. In this section we improve on the preprocessing time for polygons without holes. The previous section computes each shortest path tree, which needs time  $\mathcal{O}(n^2 \log n)$ . In polygons without holes, we can use a different technique to avoid this large computation. The routing function, the labelling of the sites and the structure of the routing table remain the same. The computation of the routing table is different. Thus, we have to prove that the preprocessing provides the same properties as in the previous section.



First of all, let  $p$  be a vertex of  $P$ . We compute its visibility polygon  $\text{vis}(p)$ . This computation provides a sequence of consecutive points  $v_0 v_1 \dots v_m$  with  $p = v_0 = v_m$ . Some points of this sequence may not be vertices of  $P$ . We assume that the sequence is sorted clockwise. Thereby, the points are sorted by increasing angle  $\alpha_j$  that is spanned by the ray  $r_0(p)$  and the edge  $\{p, v_j\}$  ( $1 \leq j \leq m-1$ ).

Let  $w_i$  be the intersection of  $r_i(p)$  and  $\text{vis}(p)$  closest to  $p$ . The corresponding edge  $e_i \subset P$  with  $w_i \in e_i \cap r_i(p)$  can be found by using the sorted sequence of vertices of  $\text{vis}(p)$  (see Figure 5).



**Fig. 5:** The boundaries of  $C_i(p)$  hit  $\partial P$  in the points  $w_{i-1}$  and  $w_i$ . The site  $s$  is the point in  $C_i(p)$  with shortest distance to  $p$ .

Now let  $C_i(p) \in \mathcal{C}(p)$  be a cone. Recall that the boundaries of  $C_i(p)$  are the rays  $r_{i-1}(p)$  and  $r_i(p)$ . The vertices related to this cone are determined in the following manner: starting from  $w_{i-1}$  we walk along the boundary of  $P$  until we meet  $w_i$  and collect all the visited vertices. Obviously, this collection forms a (possibly empty) interval, called  $I(i)$ . Next, we look for the vertex  $s$  with smallest distance to  $p$  among the points of  $I(i)$ . As before, we store the interval boundary of  $I(i)$  together with the label of the vertex  $s$  in the routing table of  $p$ . This needs  $\mathcal{O}(\log n)$  bits as well. Again, by Lemma 4.2, the size of the routing tables are  $\mathcal{O}(\varepsilon^{-1} \log n)$  bits. This result is the same as in the previous section.

With this we obtain the following lemma.

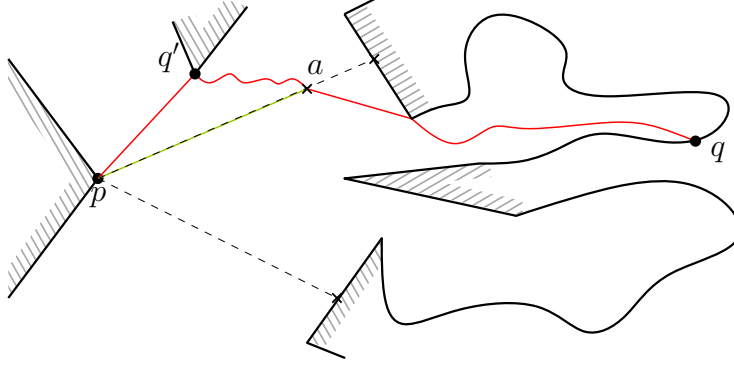
**Lemma 6.1.** *Let  $p, q$  be two vertices of  $P$  and  $(p, q')$  the first edge on the shortest path from  $p$  to  $q$ . If  $q \in I(i)$ , then  $q' \in C_i(p)$ .*

*Proof.* Suppose that  $q' \notin C_i(p)$ . Since  $q$  is in  $I(i)$ , the shortest path  $\pi$  from  $p$  to  $q$  has to cross  $\overline{pw_{i-1}}$  or  $\overline{pw_i}$  at least twice. The first intersection is  $p$  itself. Let  $a \neq p$  be the second intersection and  $\pi'$  the subpath of  $\pi$  from  $p$  to  $a$ . By the triangle inequality  $|\overline{pa}|$  is strictly smaller than the length of  $\pi'$  (see Figure 6). Thus, we can find a shortcut from  $p$  to  $a$  and hence a shorter path from  $p$  to  $q$ . This contradicts the assumption that  $\pi$  is a shortest path from  $p$  to  $q$  and finishes the proof.  $\square$

This lemma proves that the routing function behaves as our first algorithm and the correctness follows. Finally, we obtain our main theorem for simple polygons.

**Theorem 6.2.** *Let  $P$  be a simple polygon with  $n$  vertices. For any  $\varepsilon > 0$  we can preprocess  $P$  and construct a routing scheme for  $\text{VG}(P)$  with labels of size  $\mathcal{O}(\log n)$  bits and routing tables of size  $\mathcal{O}(\varepsilon^{-1} \log n)$  bits. For any two sites  $p, q \in P$ , the scheme produces a routing path with stretch  $1 + \varepsilon$ . The preprocessing time is  $\mathcal{O}(n^2 + \varepsilon^{-1}n)$ .*

*Proof.* Let  $p$  be a vertex of  $P$ . First, we compute the visibility polygon of the vertex  $p$ . This needs time  $\mathcal{O}(n^2)$ . Now, let  $L$  be the sequence of vertices of  $\text{vis}(p)$  sorted by increasing angle.



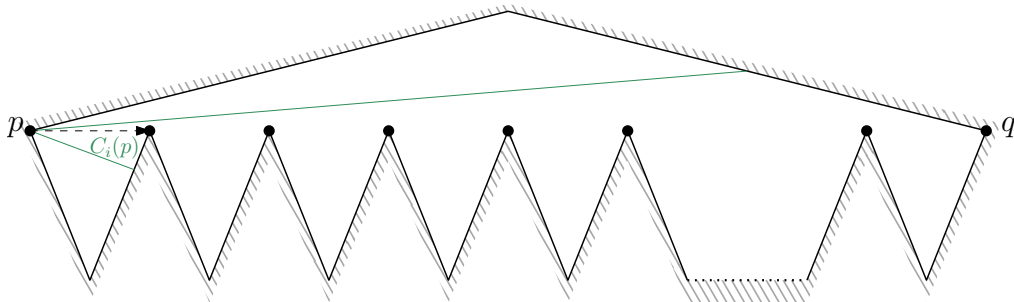
**Fig. 6:** The red line is the “shortest” path from  $p$  to  $q$  with  $q'$  as first step, whereas the green dashed line represents a shortcut from  $p$  to  $a$ .

Using  $L$ , we can find in time  $\mathcal{O}(n + \varepsilon^{-1})$  all intersecting points  $w_i$  and corresponding edges  $e_i$  of  $P$ . Thus, for each ray  $r_i(p)$ , we can find in constant time the boundaries of  $e_i$ . Finally, let  $C_i(p)$  be a cone. We can find in constant time the interval boundaries of  $I(i)$  and in  $\mathcal{O}(|I(i)|)$  time the point with the smallest distance in  $I(i)$  to  $p$ . We store the interval and the label of the point in the  $i$ -th row of the routing table of  $p$ . This step costs  $\mathcal{O}(n + \varepsilon^{-1})$  in total over all cones. We perform the same procedure for all vertices and obtain the running time  $\mathcal{O}(n^2 + \varepsilon^{-1}n)$ .  $\square$

## 7 Conclusion

We presented an efficient routing scheme for the visibility graph of a polygon with holes. The routing scheme produces a routing path whose length can be made arbitrarily close to the optimum.

There are still various open questions for the routing schemes for polygons. First of all, it would be interesting to know whether there is a routing scheme that approximates the hop-distance in polygons, where each pair of adjacent vertices has edge weight 1. Using our routing scheme we can find examples, where the stretch is  $\Omega(n)$  (see Figure 7). Moreover, it would be interesting to know whether the preprocessing time or the size of the routing table can be improved, perhaps using a recursive strategy.



**Fig. 7:** In this polygon, the hop-distance between  $p$  and  $q$  is 1, because they can see each other. Our routing scheme routes from one spire to the next. Thus, the stretch is  $\Theta(n)$ .

There is one more open question for routing schemes in general: what is the time needed by a data package to travel through the graph? In particular, it would be interesting to see how much time a data package needs at one single node. It would be a slightly different but important measure of routing schemes.

## References

- [1] Ittai Abraham and Cyril Gavoille. On approximate distance labels and routing schemes with affine stretch. In *Proc. 25th Int. Symp. Dist. Comp. (DISC)*, pages 404–415. Springer, 2011.
- [2] Takao Asano, Tetsuo Asano, Leonidas Guibas, John Hershberger, and Hiroshi Imai. Visibility of disjoint polygons. *Algorithmica*, 1(1-4):49–63, 1986.
- [3] Baruch Awerbuch, Amotz Bar-Noy, Nathan Linial, and David Peleg. Improved routing strategies with succinct tables. *J. Algorithms*, 11(3):307–341, 1990.
- [4] Reuven Bar-Yehuda and Bernard Chazelle. Triangulating disjoint Jordan chains. *Internat. J. Comput. Geom. Appl.*, 4(04):475–481, 1994.
- [5] Shiri Chechik. Compact routing schemes with improved stretch. In *Proc. ACM Symp. Princ. Dist. Comp. (PODC)*, pages 33–41. ACM, 2013.
- [6] Lenore J Cowen. Compact routing with minimum stretch. *J. Algorithms*, 38(1):170–183, 2001.
- [7] Tamar Eilam, Cyril Gavoille, and David Peleg. Compact routing schemes with low stretch factor. *J. Algorithms*, 46(2):97–114, 2003.
- [8] Pierre Fraigniaud and Cyril Gavoille. Routing in trees. In *Proc. 28th Internat. Colloq. Automata Lang. Program. (ICALP)*, pages 757–772, 2001.
- [9] Silvia Giordano and Ivan Stojmenovic. Position based routing algorithms for ad hoc networks: A taxonomy. In *Ad hoc wireless networking*, pages 103–136. Springer-Verlag, 2004.
- [10] Leonidas Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert E Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1-4):209–233, 1987.
- [11] John Hershberger and Subhash Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM J. Comput.*, 28(6):2215–2256, 1999.
- [12] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Routing in unit disk graphs. In *Proc. 12th Latin American Symp. Theoretical Inf. (LATIN)*, pages 536–548, 2016.
- [13] Sanjiv Kapoor and SN Maheshwari. Efficient algorithms for Euclidean shortest path and visibility problems with polygonal obstacles. In *Proc. 4th Annu. Sympos. Comput. Geom. (SoCG)*, pages 172–182, 1988.
- [14] Sanjiv Kapoor, SN Maheshwari, and Joseph SB Mitchell. An efficient algorithm for Euclidean shortest paths among polygonal obstacles in the plane. *Discrete Comput. Geom.*, 18(4):377–383, 1997.
- [15] Joseph SB Mitchell. A new algorithm for shortest paths among obstacles in the plane. *Annals of Mathematics and Artificial Intelligence*, 3(1):83–105, 1991.
- [16] Joseph SB Mitchell. Shortest paths among obstacles in the plane. *Internat. J. Comput. Geom. Appl.*, 6(03):309–332, 1996.
- [17] Mark H Overmars and Emo Welzl. New methods for computing visibility graphs. In *Proc. 4th Annu. Sympos. Comput. Geom. (SoCG)*, pages 164–171, 1988.
- [18] David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *J. ACM*, 36(3):510–530, 1989.
- [19] Liam Roditty and Roei Tov. New routing techniques and their applications. In *Proc. ACM Symp. Princ. Dist. Comp. (PODC)*, pages 23–32. ACM, 2015.
- [20] Liam Roditty and Roei Tov. Close to linear space routing schemes. *Distributed Computing*, 29(1):65–74, 2016.
- [21] Nicola Santoro and Ramez Khatib. Labelling and implicit routing in networks. *The Computer Journal*, 28(1):5–8, 1985.
- [22] Micha Sharir and Amir Schorr. On shortest paths in polyhedral spaces. *SIAM J. Comput.*, 15(1):193–215, 1986.

- [23] James A Storer and John H Reif. Shortest paths in the plane with polygonal obstacles. *J. ACM*, 41(5):982–1012, 1994.
- [24] Mikkell Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, 2004.
- [25] Mikkell Thorup and Uri Zwick. Compact routing schemes. In *Proc. 13th ACM Symp. Par. Algo. Arch. (SPAA)*, pages 1–10. ACM, 2001.
- [26] Mikkell Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005.
- [27] Emo Welzl. Constructing the visibility graph for  $n$ -line segments in  $\mathcal{O}(n^2)$  time. *Inform. Process. Lett.*, 20(4):167–171, 1985.
- [28] Chenyu Yan, Yang Xiang, and Feodor F Dragan. Compact and low delay routing labeling scheme for unit disk graphs. *Comput. Geom. Theory Appl.*, 45(7):305–325, 2012.
- [29] Andrew Chi-Chih Yao. On constructing minimum spanning trees in  $k$ -dimensional spaces and related problems. *SIAM J. Comput.*, 11(4):721–736, 1982.